# JerryScript Debugger

Zoltan Herczeg
zherczeg.u-szeged@partner.samsung.com

Samsung Research Hub @ University of Szeged

JerryScript Developer Meeting 2017
Szeged, Hun, Sept. 14, 2017

# Overview

- Introduction
- Communication
- Debugger Features
- Debugger C API
- Summary

# Introduction

# Introduction

- JerryScript debugger is a remote debugger
  - Server is part of the JerryScript binary
  - Client is a standalone / web application
- Communication: any reliable stream / message based protocol
  - Minimum message size is 64 bytes (excluding protocol header)
  - Currently WebSockets is used
    - supported natively by browsers
  - Could be ported to 6lowpan, serial port

# Debugger Client

- To reduce memory consumption, debugger related data is **stored** on the Client side, e.g:
  - Text of all compiled source codes
  - Breakpoint position and line info
- Client is expected to run on a desktop / server system
  - Has much more resources than JerryScript
- On-the-fly client attachment is not supported
  - Client must connect after `jerry_init()` is called

# Byte Code Management

- Byte code create / free must be synchronized
  - When a byte code is freed, there might be incoming messages which manipulates the byte code data (e.g. enable / disable breakpoints)
- Therefore it is not enough to notify the client about a byte code free, the client must also notify JerryScript that it can free the byte code memory data
  - Until that the byte code is kept in the memory
  - Garbage Collection waits until all pending byte code frees are acknowledged by the client

# Running the Debugger

- Building JerryScript with Debugger Server
  - `build.py –jerry-debugger=on --jerry-libc=off –static-link=off`
- Waiting for client connection
  - `build/bin/jerry --start-debug-server`
- Example clients in `jerry-debugger/` top level dir
  - `jerry-debugger` – python client
  - `jerry-client-ws.html` – browser based client

# Communication

# Debugger Message Format

- Allowed data types:
  - uint8_t, uint32_t, cpointer_t
- Each message starts with a type byte
- All types are tightly packed

```
/**
 * Incoming message: update (enable/disable) breakpoint status.
 */
typedef struct
{
  uint8_t type; /**< type of the message */
  uint8_t is_set_breakpoint; /**< set or clear breakpoint */
  uint8_t byte_code_cp[sizeof (jmem_cpointer_t)]; /**< byte code compressed pointer */
  uint8_t offset[sizeof (uint32_t)]; /**< breakpoint offset */
} jerry_debugger_receive_update_breakpoint_t;
```

# First Message From Server

- JERRY_DEBUGGER_CONFIGURATION
- Payload: three uint8_t values
  - max_message_size: maximum message size accepted by the server (minimum 64 bytes)
  - cpointer_size: size of compressed pointers
  - little_endian: 1 for little endian, 0 for big endian machines

# Debugger Server Modes

- Debugger starts in run mode
  - Messages defined for run mode must be accepted regardless of mode (e.g. free byte code, stop execution)
- Breakpoint mode
  - Backtrace and eval are accepted only in this mode
  - Continue, next, and step allows returning to run mode
- Client source mode
  - The client should send the JS source code
- The server notifies the client about mode changes

# Debugging Features

# High-level Client Features

- Set/clear breakpoints
- Execution control
  - step-in / next / continue
  - Finish could be implemented
- Automatic stop at exceptions can be enabled
- Backtrace
  - ECMAScript functions only
  - API to add extra items?

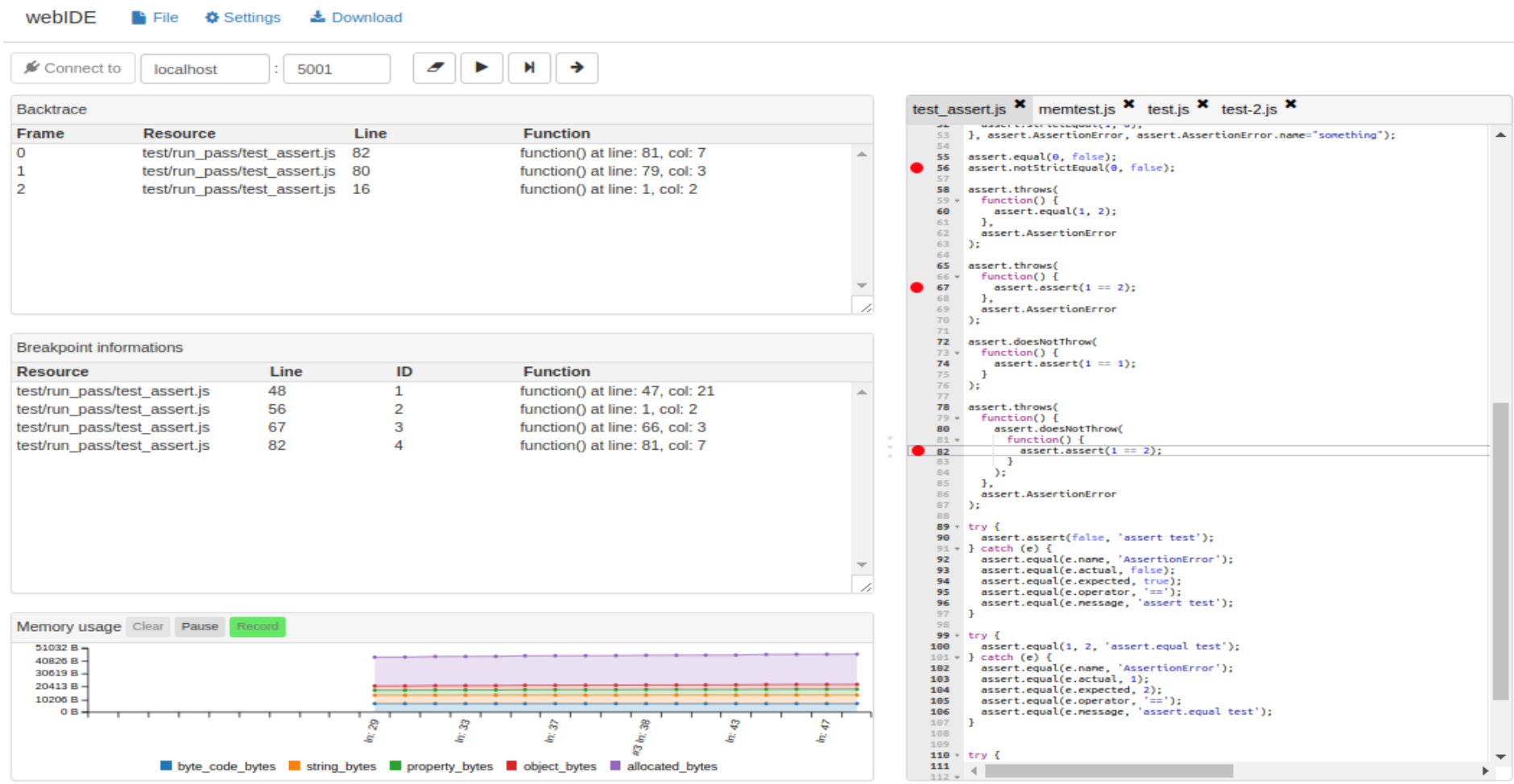
# High-level Client Features (2)

- Evaluate expressions
  - Can be used to inspect / change any variables
  - Clients can use this to gather information
    - E.g. watching variables
- Memory consumption statistics
  - Total memory consumption in bytes
  - Memory consumption of byte code data, strings, objects, properties

# Python Client Example

```
(jerry-debugger) break test.js:20
Breakpoint 1 at test.js:20
(jerry-debugger) continue
Press enter to stop JavaScript execution.
Stopped at breakpoint:1 test.js:20
Source: test.js
  18    /* Draw text on screen. */
  19
  20 > var pos = { x:36, y:48 };
  21    draw_text(pos, "Hello world!");
  22
(jerry-debugger) next
Stopped at test.js:21
Source: test.js
  19
  20    var pos = { x:36, y:48 };
  21 > draw_text(pos, "Hello world!");
  22
  23    display();
(jerry-debugger) eval pos.x + pos.y
84
(jerry-debugger) memstats
Allocated bytes: 3344
Byte code bytes: 104
String bytes: 1537
Object bytes: 192
Property bytes: 928
```

# IoT.js IDE Screenshot



# Debugger C API

# Initialize Connection

- Listening to a debugger client is requested by the application which uses the JerryScript library
  - Debugging cannot be forced onto an application
- `jerry_debugger_init (uint16_t port)`
  - Waiting for a debugger client, returns when the connection is established
- `jerry_debugger_is_connected (void)`
  - Tells whether a client is connected

# Execution Control

- `jerry_debugger_stop (void)`
  - Stop at the next possible breakpoint even if it is disabled
- `jerry_debugger_continue (void)`
  - Don't stop at disabled breakpoints
- `jerry_debugger_stop_at_breakpoint (bool enable_stop_at_breakpoint)`
  - Controls stopping at enabled breakpoints: if the argument is false the engine ignores all breakpoints
  - The application can run a JavaScript code which cannot be controlled by the debugger

# Receiving Source Code

- `jerry_debugger_wait_and_run_client_source (jerry_value_t *return_value)`
  - Receive a source code from the client and execute it
- Return values
  - JERRY_DEBUGGER_SOURCE_RECEIVE_FAILED
    - A network error is occurred, connection aborted
  - JERRY_DEBUGGER_SOURCE_RECEIVED
    - Source code is received an executed
  - JERRY_DEBUGGER_SOURCE_END
    - No source provided by the client

# Display Output at Client Side

- `jerry_debugger_send_output (jerry_char_t buffer[], jerry_size_t str_size, uint8_t type)`
  - Sends a string to the debugger client
  - JERRY_DEBUGGER_OUTPUT_OK
    - The string is a normal text
  - JERRY_DEBUGGER_OUTPUT_WARNING
    - The string is a warning message
  - JERRY_DEBUGGER_OUTPUT_ERROR
    - The string is an error message

# Summary

# Summary

- JerryScript debugger is a remote debugger
  - Debugger data is stored on the client side
- Supports the usual commands
  - Step,next,continue, backtrace, eval, etc.
- Debugging can be controlled through C API

# Future Work

- Debugging snapshots
- Abort execution and engine reset (source code receive)
- Debugging original source code after transpiling
- Community requests

# Thank you.